SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): **Ada Compiler Validation Summary Report:** Control Data Corporation, ADA/VE, Ver 1.3, CYBER 932 (host) to CYBER 932 (target), 890901S1.10147	5. TYPE OF REPORT & PERIOD COVERED: 1 Sept 89 - 12 Dec 90
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): National Institute of Standards and Technology, Gaithersburg, Maryland, USA	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS: National Institute of Standards and Technology, Gaithersburg, Maryland, USA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Ada Joint Program Office, United States Department of Defense, Washington, DC 20301-3081	12. REPORT DATE / 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office): National Institute of Standards and Technology, Gaithersburg, Maryland, USA	15. SECURITY CLASS (of this report): UNCLASSIFIED / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Control Data Corporation, ADA/VE, Ver 1.3, National Institute of Standards and Technology, CYBER 932 under NOS/VE level 727 (host & target), ACVC 1.10

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

AVF Control Number: NIST89CDC540_1.10
14 December 1989

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890901S1.10147
Control Data Corporation
ADA/VE, Ver 1.3
CYBER 932 Host and CYBER 932 Target

Completion of On-Site Testing:
1 September 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: ADA/VE, Ver 1.3

Certificate Number: 890901S1.10147

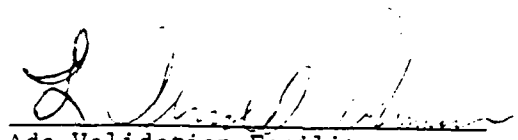
Host: CYBER 932 under NOS/VE level 727

Target: CYBER 932 under NOS/VE level 727

Testing Completed 1 September 1989 Using ACVC 1.10

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

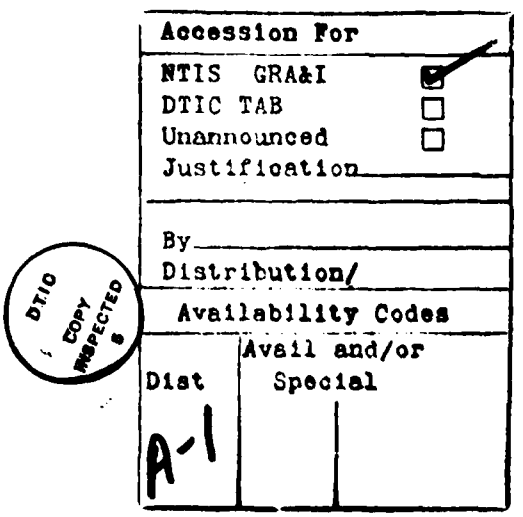


Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
MR. John Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS



CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report. The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by GEMMA Corp. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 1 September 1989 at Control Data Corporation.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institue of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada	An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer which executes the code generated by the compiler.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn	An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the

program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These

tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: ADA/VE, Ver 1.3

ACVC Version: 1.10

Certificate Number: 890901S1.10147

Host Computer:

Machine: CYBER 932

Operating System: NOS/VE level 727

Memory Size: 64 MB

Target Computer:

Machine: CYBER 932

Operating System: NOS/VE level 727

Memory Size: 64 MB

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

Predefined types.

(1) This implementation supports the additional predefined type LONG_FLOAT in the package STANDARD. (See tests B86001T..Z (7 tests).)

Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is not gradual. (See tests C45524A..Z (26 tests).)

Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

(2) CONSTRAINT_ERROR) is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

(3) CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises CONSTRAINT_ERROR when the array type is declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the array objects are declared. (See test C52104Y..)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an

implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises CONSTRAINT_ERROR when the array type is declared. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

Generics.

(1) Generic specifications and bodies cannot be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(3) Generic subprogram declarations and bodies cannot be compiled in separate compilations. (See tests CA1012A and CA2009F.)

(4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(5) Generic non-library subprogram bodies cannot be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(7) Generic library package specifications and bodies cannot be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(8) Generic non-library package bodies as subunits cannot be compiled in separate compilations. (See test CA2009C.)

Input and output.

(1) The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)

(4) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)

(5) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)

(6) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(7) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(8) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(9) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(10) Temporary sequential files are not given names and not deleted when closed. (See test CE2108A.)

(11) Temporary direct files are not given names and not deleted when closed. (See test CE2108C.)

(12) Temporary text files are not given names and not deleted when closed). (See test CE3112A.)

(13) Only one internal file can be associated with each external file for sequential files when writing or reading. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(14) Only one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(15) Only one internal file can be associated with each external file for text files when writing or reading. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 399 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 19 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 102 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	120	1127	1943	17	21	46	3274
Inapplicable	9	11	372	0	7	0	399
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	195	636	644	242	172	99	158	331	135	36	251	100	275	3274
Inapplicable	17	13	36	6	0	0	8	1	2	0	1	269	46	399
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G	B97102E	C97116A	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15B	CD2B15C	CD2D11B	CD5007B	CD5011O
CD7105A	CD7203B	CD7204B	CD7205C	CD7205D	CE2107I
CE3111C	CE3301A	CE3411B	E28005C	ED7004B	ED7005C
ED7006C	ED7006D				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 399 tests were inapplicable for the reasons indicated:

C24113I..X (16 tests) are not applicable because the length of the input file exceeds 132 characters.

The following 19 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113Y (1 test)	C35705Y (1 test)
C35706Y (1 test)	C35707Y (1 test)
C35708Y (1 test)	C35802Y..Z (2 tests)
C45241Y (1 test)	C45321Y (1 test)
C45421Y (1 test)	C45521Y..Z (2 tests)
C45524Y..Z (2 tests)	C45621Y..Z (2 tests)
C45641Y (1 test)	C46012Y..Z (2 tests)

The following 170 tests are not applicable because 'SIZE representation clauses are not supported:

A39005B C87B62A	CD1009A..I (9 tests)
CD1009O..Q (3 tests)	CD1C03A CD1C04A
CD2A21A..E (5 tests)	CD2A22A..J (10 tests)
CD2A23A..E (5 tests)	CD2A24A..J (10 tests)
CD2A31A..D (4 tests)	CD2A32A..J (10 tests)
CD2A41A..E (5 tests)	CD2A42A..J (10 tests)
CD2A51A..E (5 tests)	CD2A52A..D (4 tests)
CD2A52G..J (4 tests)	CD2A53A..E (5 tests)
CD2A54A..D (4 tests)	CD2A54G..J (4 tests)
CD2A61A..L (12 tests)	CD2A62A..C (3 tests)
CD2A64A..D (4 tests)	CD2A65A..D (4 tests)
CD2A71A..D (4 tests)	CD2A72A..D (4 tests)
CD2A74A..D (4 testS)	CD2A75A..D (4 tests)
CD2A81A..E (5 tests)	CD2A81F
CD2A83A..C (3 tests)	CD2A83E..F (2 tests)
CD2A84B..I (8 tests)	CD2A84K..L (2 tests)
CD2A87A	CD2A91A..E (5 tests)
ED2A26A ED2A56A	ED2A86A

The following 7 tests are not supported because 'SMALL representation clauses are not supported:

A39005E C87B62C CD1009L CD1C03F CD1C04C CD2D11A CD2D13A

C35508I, C35508J, C35508M, and C35508N are not applicable because they include enumeration representation clauses for BOOLEAN types in which the representation values are other than (FALSE => 0, TRUE => 1). Under the terms of AI-00325, this implementation is not required to support such representation clauses.

C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

The following 16 tests are not applicable because this implementation does not support a predefined type SHORT_INTEGER:

C45231B C45304B C45502B C45503B C45504B
C45504E C45611B C45613B C45614B C45631B
C45632B B52004E C55B07B B55B09D B86001V
CD7101E

The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

C45231C C45304C C45502C C45503C C45504C
C45504F C45611C C45613C C45614C C45631C
C45632C B52004D C55B07A B55B09C B86001W
CD7101F

C4A013B is not applicable because the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

B86001X, C45231D, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

The following 76 tests are not applicable because, for this implementation, type SYSTEM.ADDRESS is a limited private type:

CD5003B..I (8 tests) CD5011A..I (9 tests)
CD5011K..M (4 tests) CD5011Q..S (3 tests)
CD5012A..J (10 tests) CD5012L..M (2 tests)
CD5013A..I (9 tests) CD5013K..O (5 tests)
CD5013R..S (2 tests) CD5014A..O (15 tests)
CD5014R..Z (9 tests)

C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

CA2009C is not applicable because this implementation does not permit compilation of generic non-library package bodies as subunits in separate files from their stubs.

CA2009F is not applicable because this implementation does not permit compilation of generic non-library subprogram bodies as subunits in separate files from their stubs.

BC3204C is not applicable because this implementation does not permit compilation of generic library package bodies in files apart from their specifications.

The following 17 tests are not applicable because no representation clauses may be given for a derived type:

AD1C04D	AD3015C	AD3015F	AD3015H	AD3015K
CD3015A	CD3015B	CD3015D	CD3015E	CD3015G
CD3015I	CD3015J	CD3015L	CD4051A..D (4 tests)	

AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

AE2101H, EE2401D and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

CE2105A is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for SEQUENTIAL_IO.

CE2105B is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for DIRECT_IO.

CE2107A..H (8 tests), CE2107L, CE2110B, CE2110D, CE2111D and CE2111H are not applicable because multiple internal files cannot be associated with the same external file for sequential files or direct files. The proper exception is raised when multiple access is attempted.

CE3102F is inapplicable because text file RESET is supported by this implementation.

CE3102G is inapplicable because text file DELETE is supported by this implementation.

CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

CE3102J is inapplicable because text file OPEN with IN_FILE mode is supported by this implementation.

CE3102K is inapplicable because text file OPEN with OUT_FILE mode is not supported by this implementation.

CE3109A is inapplicable because text file CREATE with IN_FILE mode is not supported by this implementation.

CE3111A..B (2 tests), CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal files cannot be associated with the same external file for text files. The proper exception is raised when multiple access is attempted.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 102 tests.

A STORAGE_SIZE clause was added to the following tests:

C34007A	C34007D	C34007G	C34007J	C34007M	C34007P
C34007S	C87B26B				

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B26001A	B26002A	B26005A	B28001D	B28003A
B29001A	B2A003A	B2A003C	B2A003D	B33102A	B33102B
B33102C	B33102D	B33102E	B33301B	B35101A	B37106A
B37301B	B37302A	B38003A	B38003B	B38009A	B38009B
B51001A	B53009A	B54A01C	B54A01H	B54A01J	B54A01K
B55A01A	B55A01C	B55A01H	B55A01I	B55A01N	B55A01O
B56001C	B56001E	B56001F	B61001C	B61001D	B61001E
B61001F	B61001H	B61001I	B61001M	B61001R	B61001W
B66001C	B67001H	B67001J	B67001K	B91001A	B91001C
B91002A	B91002B	B91002C	B91002D	B91002E	B91002F
B91002G	B91002H	B91002I	B91002J	B91002K	B91002L
B95030A	B95061A	B95061F	B95061G	B95077A	B97101A
B97101E	B97101G	B97101H	B97103E	B97104G	BA1101B
BC1008A	BC1016A	BC1016B	BC1109A	BC1109C	BC1109D
BC1202A	BC1202B	BC1202E	BC1202F	BC1202G	BC2001C
BC2001D	BC2001E	BC2004A	BC3013A		

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the ADA/VE compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the ADA/VE compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	CYBER 932
Host operating system:	NOS/VE level 727
Target computer:	CYBER 932
Target operating system:	NOS/VE level 727

A tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precision was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on site. Tests requiring modifications during the prevalidation testing were included in their modified form on the tape.

TEST INFORMATION

The contents of the tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the CYBER 932.

The compiler was tested using command scripts provided by Control Data Corporation and reviewed by the validation team. The compiler was tested using all default option settings. See Appendix E for a complete listing of the compiler options for this implementation.

Tests were compiled, linked, and executed (as appropriate) using one host computer and one target computer. Test output, compilation listings, and job logs were captured on tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Control Data Corporation and was completed on 1 September 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

Control Data Corporation has submitted the following Declaration of Conformance concerning the ADA/VE compiler.

Appendix A

DECLARATION OF CONFORMANCE (SAMPLE)

Compiler Implementer: Control Data Corporation
Ada Validation Facility: NIST/NCSL
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name: ADA/VE Version: 1.3
Host Architecture - ISA: CYBER 932 OS&VER #: NOS/VE level 727
Target Architecture - ISA: same OS&VER #: same

Derived Compiler Registration

Derived Compiler Name: _____ Version: _____
Host Architecture - ISA: _____ OS&VER #: _____
Target Architecture - ISA: _____ OS&VER #: _____

Implementer's Declaration

I, the undersigned, representing Control Data Corporation (herein referred to as CDC) have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that CDC is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

_____ 7/13/89

Owner's Declaration

I, the undersigned, representing CDC take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compiler(s) and concur with the contents.

_____ 7/13/89

This document is part of the Validation Summary Report (VSR). Appendix A, for initial validations and must be submitted for each derived compiler registration during or subsequent to initial validation.

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the ADA/VE, ver 1.3 compiler, as described in this Appendix, are provided by Control Data Corporation. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

 type INTEGER is range -9223372036854775808 .. 9223372036854775807;

 type FLOAT is digits 13 range -16#7.FFFF_FFFF_FFF8E1023 ..
                    16#7.FFFF_FFFF_FFF8E1023;
 type LONG_FLOAT is digits 28 range -
       -16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023 ..
        16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023;

 type DURATION is delta 1.E-3 range -8.589934591999E09 ..
                       8.589934591999E09;

...

end STANDARD;
```

Implementation-Dependent Characteristics F

This appendix summarizes the implementation-dependent characteristics of NOS/VE Ada by listing the following:

- NOS/VE Ada pragmas
- NOS/VE Ada attributes
- Specification of the package SYSTEM
- Restrictions on representation clauses
- Conventions for implementation-generated names denoting implementation-dependent components
- Interpretation of expressions appearing in address clauses
- Restrictions on unchecked type conversions
- Implementation-dependent characteristics of input-output packages
- Other implementation characteristics

Shading is not used in this appendix.

F.1 NOS/VE Ada Pragmas

NOS/VE Ada does not support all the pragmas required by the ANSI standard for Ada. It does not provide any implementation defined pragmas.

NOS/VE Ada supports the following pragmas as described in the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except as shown below:

- INLINE

  This pragma causes inline expansion of a subroutine except as described in Annex B of this manual. See 10.6 to see how to use INLINE to create faster object code.

- INTERFACE

  This pragma is supported for FORTRAN, CYBIL, and the NOS/VE MATH LIBRARY, as discussed in 13.9.1, 13.9.2, and 13.9.3, respectively.

- PACK

  Objects of the given type are packed into the nearest 2**n bits.

- SHARED

  This pragma is not allowed for the following types of variables:

  - Variables of type LONG_FLOAT
  - Variables of subtype of type LONG_FLOAT
  - Variables of a type derived from type LONG_FLOAT
  - Variables of a subtype derived from type LONG_FLOAT

- SUPPRESS

  NOS/VE supports this pragma, but it is not possible to restrict the check suppression to a specific object or type.

NOS/VE Ada does not support the following pragmas:

- CONTROLLED
- MEMORY_SIZE
- OPTIMIZE
- STORAGE_UNIT
- SYSTEM_NAME

F.2 NOS/VE Ada Attributes

NOS/VE Ada supports all the attributes required by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language. It does not provide any implementation defined attributes. The NOS/VE implementation of the P'ADDRESS attribute returns the prefix P, the 48-bit program virtual address (PVA) right justified within a 64-bit variable of the predefined type INTEGER.

F.3 Specification of the Package SYSTEM

```
package SYSTEM is

   type ADDRESS is access INTEGER;

   type NAME is (CYBER180);

   SYSTEM_NAME : constant NAME := CYBER180;

   STORAGE_UNIT : constant := 64; -- 64-bit machine

   MIN_INT : constant := -9_223_372_036_854_775_808; --(-2**63)

   MAX_INT : constant := 9_223_372_036_854_775_807; --(2**63)-1

   MAX_DIGITS : constant := 28;
   MAX_MANTISSA : constant := 63;
   FINE_DELTA : constant := 2#1.0#E-63; -- 2**(-63)
   TICK : constant := 0.001;

   subtype PRIORITY is INTEGER range 0 .. 0;

end SYSTEM;
```

F.4 Restrictions on Representation Clauses

NOS/VE Ada implements representation clauses as required by the ANSI standard for Ada. It does not allow representation clauses for a derived type.

NOS/VE Ada supports the type representation clauses with some restrictions:

- Length clauses
- Enumeration representation clauses
- Record representation clauses

NOS/VE Ada does not support address clauses or interrupts.

F.4.1 Length Clauses

NOS/VE Ada supports the attributes in the length clauses as follows:

- T'SIZE
  Not supported
- T'STORAGE_SIZE (collection size)
  Supported
- T'STORAGE_SIZE (task activation size)
  Supported

- T'SMALL

  Not supported. The compiler always chooses for SMALL the largest power of 2 not greater than the delta in the fixed accuracy definition of the first named subtype of a fixed point type.

  For example, NOS/VE Ada uses the declaration:

  ```
  type ADA_FIXED is delta 0.05 range 1.00 .. 3.00;
  ```

  to set the ADA_FIXED'SMALL attribute to 0.03125 (2^{-5}, the largest power of 2 not greater than **delta**, 0.05).

F.4.2 Enumeration Clauses

In NOS/VE Ada enumeration representation clauses, the internal codes must be in the range of the predefined type INTEGER.

F.4.3 Record Representation Clauses

NOS/VE Ada implements record representation clauses as required by the ANSI language definition. It does not support alignment clauses in record representation clauses.

The component clause of a record representation clause gives the storage place of a component of the record, by providing the following pieces of data:

- The name gives the name of the record component.
- The simple expression following the reserved word AT gives the address in storage units, relative to the beginning of the record, of the storage unit where the component starts.
- The range in the component clause gives the bit positions, relative to that starting storage unit, occupied by the record component.

NOS/VE Ada supports the range for only those record components of discrete type: integer or enumeration. The range must specify 16 bits or fewer. Furthermore, if the range specifies 8 or more bits, then the range must be a multiple of 8 bits.

F.5 Conventions for Implementation-Generated Names Denoting Implementation-Dependent Components

NOS/VE Ada does not support implementation-dependent names to be used in record representation clauses.

F.6 Interpretation of Expressions Appearing in Address Clauses

NOS/VE Ada does not support address clauses and interrupts.

F.7 Restrictions on Unchecked Type Conversions

NOS/VE Ada allows unchecked conversions when objects of the source and target types have the same size.

F.8 Input-Output Packages

The discussion of NOS/VE Ada implementation of input-output packages includes the following:

- External files and file objects
- NOS/VE Ada implemented exceptions for input-output errors
- Low level input-output

F.8.1 External Files and File Objects

NOS/VE Ada can process files created by another language processor as long as the data types and file structures are compatible.

NOS/VE Ada characterizes an external file as described in the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except as noted below:

- Name

  (see 14.2.1)

- Form

  Form strings have no effect on file processing in NOS/VE Ada. The FORM function returns the string you provided in the CREATE or OPEN procedure call.

NOS/VE Ada supports the following kinds of external files:

- Sequential access files (see 14.1)
- Direct access files (see 14.1)
- Text input-output files (see 14.3)

F.8.2 Exceptions for Input-Output Errors

NOS/VE Ada raises the following language-defined exceptions for error conditions detected during input-output operations:

- DATA_ERROR
- DEVICE_ERROR
- END_ERROR
- NAME_ERROR
- USE_ERROR

The DATA_ERROR is raised when:

- An attempt is made to read a direct file with an index that has not been defined.
- A check reveals that the sizes of the records read from a file do not match the sizes of the Ada variables. NOS/VE Ada performs this check except in those few instances where it is too complicated to do so. NOS/VE Ada omits the check in these cases, as permitted by the ANSI Ada language definition (see 14.2.2).

The END_ERROR is raised when:

An attempt is made to read a file beyond end_of_file.

The NAME_ERROR is raised when:

The file name used in CREATE or OPEN is not a valid NOS/VE file name.

The USE_ERROR is raised when:

- The TEXT_IO package tries to process a file without the LIST attribute (see 14.3).
- The NAME function tries to operate on a temporary file (see 14.2.1).
- A file overflow condition exists.
- An attempt is made to delete an external direct file with multiple accesses while more than one instance of open is still active. The file remains open and the position is unchanged.
- An attempt is made to create a sequential, text, or direct file of mode IN_FILE.
- An attempt is made to create a file that currently exists.
- An attempt is made to process a text file that is longer than 511 characters.
- An attempt is made to set the page length to other than unbounded for a non-list text file.
- An attempt is made to set line length to less than the current line length on an OUT non-list text file.
- An attempt is made to issue a NEW_PAGE for a non-list text file.

F.8.3 Low Level Input-Output

As permitted by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language Manual, NOS/VE Ada does not support the LOW_LEVEL_IO package.

F.9 Other Implementation-Dependent Characteristics

The other implementation-dependent characteristics of NOS/VE Ada are discussed as follows:

- Implementation features
- Entity types
- Tasking
- Interface to other languages
- Command interfaces
- Values of data attributes

F.9.1 Implementation Features

The NOS/VE Ada implementation features are listed as follows:

- Predefined types
- Basic types
- Compiler
- Definition of a main program
- TIME type
- Machine code insertions

F.9.1.1 Predefined Types

NOS/VE Ada implements all the predefined types required by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except:

- LONG_INTEGER
- SHORT_FLOAT
- SHORT_INTEGER

F.9.1.2 Basic Types

The sizes of the basic types are as follows:

Type	Size (bytes)
ENUMERATION	8
FIXED	8
FLOAT	8
INTEGER	8
LONG_FLOAT	16
TASK	8

In NOS/VE Ada, the enumeration type can be a boolean type or a character type as well as a user defined enumeration type.

F.9.1.3 Compiler

NOS/VE Ada provides an ANSI standard Ada compiler. The discussion of this compiler contains two types of information:

- The physical realization of the compiler
- Performance hints

F.9.1.3.1 Physical Realization of the Compiler

The NOS/VE Ada compiler supports the following:

- Source code lines up to 132 characters long
- Up to 100 static levels of nesting of blocks and/or subprograms
- External files up to one segment, 2**31-1 bytes, in length
- Compiling a generic package body that is a library unit, if both the body and the declaration of its generic package are in the same file input to the NOS/VE Ada compiler
- Compiling as a subunit a generic package body that is a secondary unit, if both the body and the declaration of its generic package are in the same file input to the NOS/VE Ada compiler
- Compiling as a subunit a generic subprogram body that is a secondary unit, if both the body and the declaration of its generic subprogram are in the same file input to the NOS/VE Ada compiler

For Better Performance

The compiler throughput improves when you submit multiple compilation units. However, if the number of compilation units grows over a certain limit, for example 50 small compilation units of about 50 lines each, or if the first compilation units are large, the throughput actually degrades.

Using the INLINE pragma, where applicable, results in faster object code by avoiding the call/return instructions.

F.9.1.4 Definition of a Main Program

NOS/VE Ada requires that the main program be a procedure without parameters. The name of a compilation unit used as a main program must follow SCL naming standards. The name can be up to 31 characters in length and must be a valid SCL name and a valid Ada identifier. Any naming error is detected at link time only. For more details about naming the main program, see the Ada for NOS/VE Usage manual.

F.9.1.5 TIME Type

NOS/VE Ada defines the type TIME as an integer representing the Julian date in milliseconds.

F.9.1.6 Machine Code Insertions

As permitted by the ANSI Ada language definition, NOS/VE Ada does not support machine code insertions.

F.9.2 Entity Types

This discussion contains information on:

- Array types
- Record types
- Access types

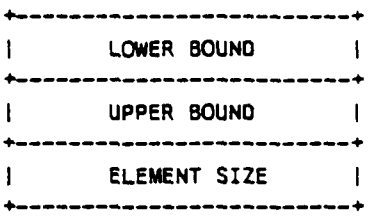
F.9.2.1 Array Types

Arrays are stored row wise, that is, the last index changes the fastest.

An array has a type descriptor that NOS/VE Ada uses when the array is one of the following:

- A component of a record with discriminants
- Passed as a parameter
- Created by an allocator

For each index, NOS/VE Ada builds the following triplet:



For multi-dimension arrays, NOS/VE Ada allocates the triplets consecutively.

Element size is expressed in number of storage units (64-bit words). If the array is packed, the element size is expressed in number of bits and represented by a negative value.

NOS/VE Ada strings are packed arrays of characters. Each component of the array is an 8-bit (1-byte) character. Packed arrays of booleans use 1 bit per component and are left justified. Arrays of integers or enumeration variables can also be packed. Each component uses n bits. Thus, the integer or enumeration subtype is in the range -2**n .. (2**n)-1.

Note that all objects start on a storage unit (64-bit word) boundary.

At run-time when NOS/VE Ada elaborates an array definition, the amount of available space remaining either on the stack or in the heap limits the maximum size of the array (see 3.6).

F.9.2.2 Record Types

At run-time when NOS/VE Ada elaborates a record definition, the amount of available space remaining either on the stack or in the heap limits the maximum size of the record (see 3.7).

NOS/VE Ada raises the exception STORAGE_ERROR at run-time when the size of an elaborated object exceeds the amount of available space.

The rest of this discussion on how records are stored includes:

- Simple record types (without discriminants)
- Record types with discriminants

F.9.2.2.1 Simple Record Types (Without Discriminants)

In the absence of representation clauses, each record component is word aligned. NOS/VE Ada stores the record components in the order they are declared.

A fixed size array (lower and upper bounds are constants) is stored within the record. Otherwise, the array is stored elsewhere in the heap, and is replaced by a pointer to the array value (first element of the array) in the record.

F.9.2.2.2 Record Types With Discriminants

The discriminants are stored first, followed by all the other components as described for simple records.

If a record component is an array with index values that depend on the value of the discriminant(s), the array and its descriptor are both allocated on the heap. They are replaced by a pair of pointers in the record. One points to the array value and the other points to the array descriptor.

F.9.2.3 Access Types

Objects of access type are simply 6-byte pointers, left justified within a word, to the accessed data contained in some allocated area in the heap. If the accessed data is of type array or packed array, the allocated area also contains the address of the array descriptor in front of the data.

F.9.3 Tasking

NOS/VE Ada supports tasking by running all Ada tasks as NOS/VE concurrent procedures. The standard NOS/VE scheduling mechanism schedules the Ada tasks. NOS/VE Ada can run at least 7 active concurrent tasks. If you wish to run more than 7 active concurrent tasks, ask your site administrator to change your site's TASK_LIMIT accordingly. The NOS/VE ANSI standard Ada compiler does not detect task termination. See the Ada for NOS/VE Usage manual for more description of NOS/VE Ada tasking.

F.9.4 Interface to Other Languages

NOS/VE Ada supports calls to CYBIL and FORTRAN subprograms and to NOS/VE MATH_LIBRARY subroutines with the following restrictions:

- CYBIL interface
  (See 13.9.1 and the Ada for NOS/VE Usage manual.)
- FORTRAN interface
  (See 13.9.2 and the Ada for NOS/VE Usage manual.)
- MATH_LIBRARY interface
  (See 13.9.3 and the Ada for NOS/VE Usage manual.)

F.9.5 Command Interfaces

The discussion of the command interfaces implemented by NOS/VE Ada includes:

- Program Library Utility commands
- Compiler command
- Linker command
- Execution commands

NOS/VE Ada commands use the syntax and language elements for parameters described in the SCL Language Definition manual.

F.9.5.1 Program Library Utility Commands

NOS/VE Ada provides a hierarchically structured (tree structure) Program Library to fulfill the ANSI Ada language definition requirements. A node (sublibrary) in the tree can reference up to 4096 compilation units or other sublibraries. The Ada for NOS/VE Usage manual contains a detailed discussion of the NOS/VE Ada implementation of the Program Library.

F.9.5.2 Compiler Command

The NOS/VE Ada compiler can compile an ANSI standard Ada program on NOS/VE. See the Ada for NOS/VE Usage manual to learn how to use the NOS/VE Ada compiler command.

F.9.5.3 Linker Command

NOS/VE Ada provides a linker to link an Ada program.

The Ada for NOS/VE Usage manual tells more about the linker command and its use.

F.9.5.4 Execution Command

NOS/VE Ada for NOS/VE provides several ways to load and execute an Ada program. They are described in the following manuals:

- Ada for NOS/VE Usage
- CYBIL System Interface Usage
- SCL Object Code Management Usage

F.9.6 Values of Data Attributes

The package STANDARD contains the declaration of the following predefined types and their attributes:

- Integer (INTEGER)
- Floating point (FLOAT)
- Long floating point (LONG_FLOAT)
- Fixed point (FIXED)
- Duration (DURATION)

F.9.6.1 Values of Integer Attributes

Attribute	Value
FIRST	-9_223_372_036_854_775_808
LAST	9_223_372_036_854_775_807
SIZE	64

F.9.6.2 Values of Floating Point Attributes

Attribute	Value
DIGITS	13
EPSILON	5.6_843_419_961#E-14
FIRST	-16#7.FFFF_FFFF_FFF8#E1023
LAST	16#7.FFFF_FFFF_FFF8#E1023
MACHINE_EMAX	4095
MACHINE_EMIN	-4096
MACHINE_MANTISSA	48
MACHINE_OVERFLOWS	TRUE
MACHINE_RADIX	2
MACHINE_ROUNDS	FALSE
SAFE_EMAX	4095
SAFE_LARGE	16#7.FFFF_FFFF_FFC#E1023
SAFE_SMALL	16#1.0#E-1024
SIZE	64

F.9.6.3 Values of Long Floating Point Attributes

Attribute	Value
DIGITS	28
FIRST	-16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023
LAST	16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023
MACHINE_EMAX	4095
MACHINE_EMIN	-4096
MACHINE_MANTISSA	96
MACHINE_OVERFLOWS	TRUE
MACHINE_RADIX	2
MACHINE_ROUNDS	FALSE
SAFE_EMAX	4095
SAFE_LARGE	16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF#E1023
SAFE_SMALL	16#1.0#E-1024
SIZE	128

F.9.6.4 Values of Fixed Point Attributes

Attribute	Value
FIRST	-9_223_372_036_854_775_808 (SMALL=1)
LAST	9_223_372_036_854_775_807 (SMALL=1)
MACHINE_ROUNDS	FALSE
MACHINE_OVERFLOWS	TRUE
SIZE	64

F.9.6.5 Values of Duration Attributes

Attribute	Value
FIRST	-6_279_897_600.0
LAST	6_279_897_600.0
DELTA	0.001
MACHINE_ROUNDS	FALSE
MACHINE_OVERFLOWS	TRUE
SMALL	2#1.0#E-10
SMALL_POWER	-10
SIZE	64

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Annex I of the Pre-Validation Letter

Parameters for TST-tests

This annex describes the changes CDC has made in the subset of the ACVC 1.10 validation suite with the type ".TST". The TST-tests are parameterized to accept certain implementation dependent values.

The following values have been substituted for the parameters in the TST programs:

```
$BIG_ID1                                -> <131 X "A">1
$BIG_ID2                                -> <131 X "A">2
$BIG_ID3                                -> <65 X "A">3<66 X "A">
$BIG_ID4                                -> <65 X "A">4<66 X "A">
$BIG_INT_LIT                            -> <129 X "0">298
$BIG_REAL_LIT                           -> <126 X "0">69.0E1
$BIG_STRING1                            -> <66 X "A">
$BIG_STRING2                            -> <65 X "A">1
$BLANKS                                 -> <112 X " ">
$COUNT_LAST                             -> 9223372036854775807
$FIELD_LAST                             -> 67
$FILE_NAME_WITH_BAD_CHARACTERS          -> BAD_CHARS^#.%!X
$FILE_NAME_WITH_WILD_CARD_CHARACTERS    -> This_File_Name_Has_
                                           To_Be_Too_Long_Wild_
                                           Card_Char_Do_Not_Exist
$GREATER_THAN_DURATION                  -> 100000000.0
$GREATER_THAN_DURATION_BASE_LAST        -> 7000000000.0
$ILLEGAL_EXTERNAL_FILE_NAME1            -> BADCHARS^@.~!
$ILLEGAL_EXTERNAL_FILE_NAME2            -> MUCH_TOO_LONG_NAME_
                                           FOR_A_VE_FILE
$INTEGER_FIRST                          -> -9223372036854775808
$INTEGER_LAST                           -> 9223372036854775807
$INTEGER_LAST_PLUS_1                    -> 9223372036854775808
$LESS_THAN_DURATION                     -> -100000000.0
$LESS_THAN_DURATION_BASE_FIRST          -> -7000000000.0
$MAX_DIGITS                             -> 28
$MAX_IN_LEN                             -> 132
$MAX_INT                                -> 9223372036854775807
$MAX_INT_PLUS_1                         -> 9223372036854775808
$MAX_LEN_INT_BASED_LITERAL              -> 2#<127 X "0">11#
$MAX_LEN_REAL_BASED_LITERAL             -> 16#<125 X "0">F.E#
$MAX_STRING_LITERAL                     -> "<130 X "A">"
$MIN_INT                                -> -9223372036854775808
$NAME                                   -> DOES_NOT_EXIST
$NEG_BASED_INT                          -> 8#1<20 X "7">6#
$NON_ASCII_CHAR_TYPE                    -> (NON-NULL)
```

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G
This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E
This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A
This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B
This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D
This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84N & M, & CD50110
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B
This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B
This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A
This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D
This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I
This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C
This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A
This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B
This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C
This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

Control Data Corporation

Compiler: ADA/VE, Ver 1.3

ACVC Version: 1.10

Annex VII of the Prevalidation Letter

Compiler Options Used

This annex describes the compiler options used in compiling the ACVC 1.10 test suite.

The following parameters were used for compiling the ACVC test suite:

- Name of the source text file
- Name of the program library that must always be specified
- Name of the source listing file
- Name of the error file

For all other options, the default values were used.

The following parameters were used for linking the ACVC tests:

- Name of the main program
- Name of the program library that must always been specified

For all other options, the default values were used.

The ACVC tests were loaded and executed through reference to the binary file $LOCAL.LGO produced by default by the Linker. This is a standard CYBER 180 feature.